N94-18376

Asynchronous Sequential Circuit Design Using Pass Transistor Iterative Logic Arrays

M. N. Liu, G. K. Maki and S. R. Whitaker
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract – **The Iterative Logic Array (ILA) is introduced as a new architecture for asynchronous sequential circuits. This is the first ILA architecture for sequential circuits reported in the literature. The ILA architecture produces a very regular circuit structure. Moreover, it is immune to both 1-1 and 0-0 crossovers and is free of hazards. This paper also presents a new critical race free STT state assignment which produces a simple form of design equations that greatly simplifies the ILA realizations.**

1 Introduction

A major goal of modern Very Large Scale Integrated (VLSI) design is to produce a structure that consists of similar, if not identical, modules. With such a structure only one module needs to be designed and then can be replicated to realize the final circuit. Very few design procedures have been advanced for sequential circuits that produce structures that are easily realized for VLSI implementations.

This paper introduces an Iterative Logic Array (ILA) architecture for the realization of asynchronous sequential circuits. With the ILA architecture, an asynchronous sequential circuit can be built in a very regular form with a single type of ILA module as a building block. Furthermore, the ILA asynchronous circuits have some unique features, such as immunity to both 1-1 overlapping and 0-0 crossing, tolerant of function hazards and immunity to input bounce. The fundamental mode operation is still required in the ILA asynchronous sequential circuits.

To further simplify the circuit of ILA module, a new state assignment has been developed to generate a new form of design equations (the simple term equation). In the simple term design equations, each coefficient is simply a state variable or a constant, instead of the random sum-of-product expression found in a traditional design equation. That simplifies the basic ILA module to a simple pass logic multiplexer.

2 ILA Architecture

Iterative Logic Arrays (ILA) has been described in the literature for quite some time [1,2]. An ILA circuit consists of an array of identical cells. Generally, as shown in Figure 1, each ILA cell contains two sets of input signals. One set of inputs are applied in parallel, while

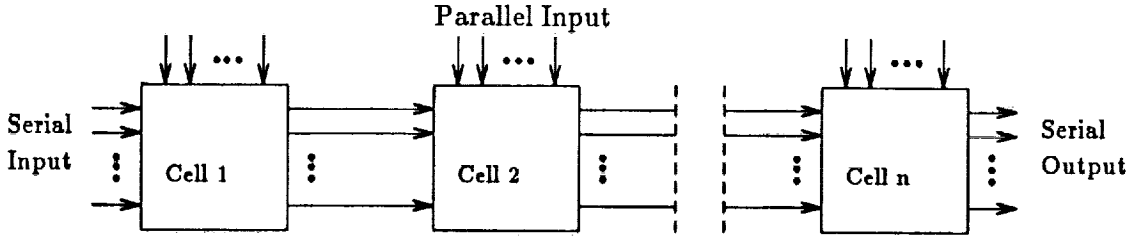

Figure 1: A slice of ILA circuit

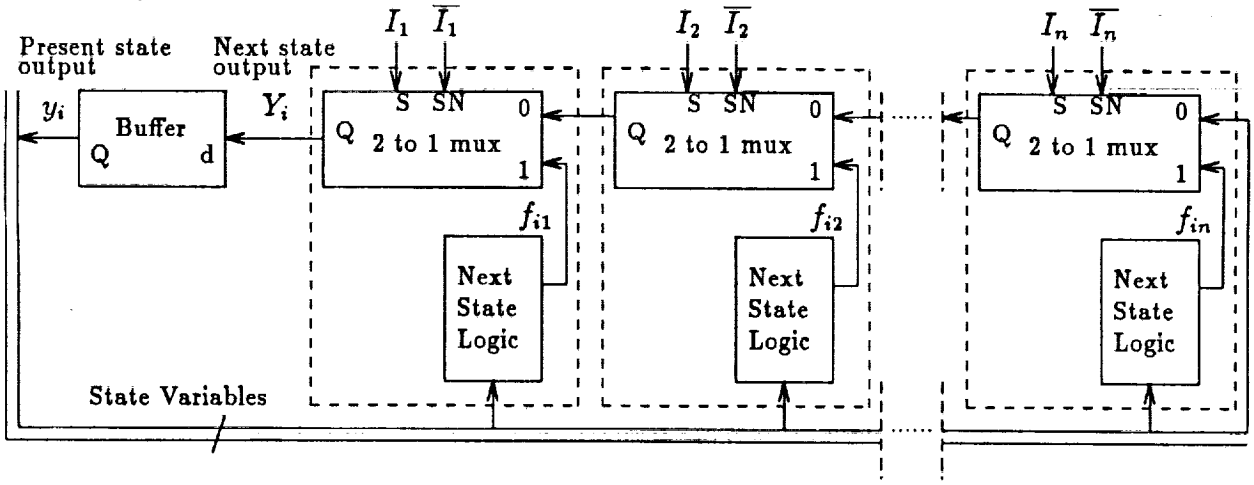

Figure 2: The overall ILA architecture

the other set of inputs are driven by adjacent cells. Signals normally propagate in only one direction between cells, and outputs are derived only from the serial outputs of the last cell. This paper presents an ILA architecture for sequential circuits in which the next state of each state variable is generated by a slice of concatenated ILA cells. A sequential network is then constructed by placing the ILA slices side by side. The function of a flow table is implemented by interconnecting ILA cells and the input states.

The basic cell of an ILA sequential network consists of a 2-to-1 multiplexer (MUX) and a next state forming logic. A MUX cell has a select line S, its complement $\overline{S}$ and two data inputs I_0 and I_1, such that $Q = S * I_1 + \overline{S} * I_0$.

The simplest way to implement the MUX function is to use a pass transistor circuit. Basically, the pass transistor MUX, excluding level restoration logic, is a module of two pass transistors, which functions as two simple switches. Design considerations, such as level restoration, are assumed to be handled by the output buffers. The circuit design considerations have been discussed in [3,4,5].

The overall architecture for the ILA sequential circuit is shown in Figure 2 which implements the next state equation

$$Y_i = f_{i1}I_1 + f_{i2}I_2 + \cdots + f_{in}I_n. \quad (1)$$

Theorem 1 *The architecture depicted in Figure 2 is a proper model for an asynchronous sequential circuit.*

Proof: It is assumed that an STT assignment is used and the logic equations are defined in Equation 1. Let the present input be I_p, which means that only MUX cell p passes the next logic state f_{ip} to the buffer. Therefore, $Y_i = f_{ip}I_p$ for each y_i. When input I_q is present ($I_p = 0, I_q = 1$), the only MUX that passes next logic state is cell q. Therefore $Y_i = f_{iq}I_q$ for each y_i. Clearly, the architecture realizes Equation 1. □

The present state is depicted by the state variables and is fed back to each ILA cell. The logic for each state variable Y_i consists of n ILA cells as defined in Figure 2. The set of n cells is described as a slice that realizes a next state variable. Referring to Figure 2, all ILA cells that are driven by the same input state I_i belong to the same level. With n input states and m state variables, there are n levels of ILA cells and m slices.

3 Design Procedure for Asynchronous ILA Sequential Network

As a general architecture, the ILA architecture can be used to realize the asynchronous design equations of any STT assignment. This section compares the ILA circuits for traditional STT assignments and proposes a new state assignment which minimizes the next state forming logic in the ILA cell.

3.1 Simple Term Design Equations

The set of next state equations provides a mathematical model of a sequential circuit. For example, the design equations using Liu's assignment for Table 1 are:

$$Y_1 = y_1I_1 + (y_1\overline{y_2} + y_2y_4)I_2 + \overline{y_3}I_3$$
$$Y_2 = y_1I_1 + y_2I_2 + y_3I_3$$
$$Y_3 = 0 + \overline{y_2}I_2 + y_3I_3$$
$$Y_4 = 0 + y_2y_4I_2 + y_3I_3$$

If an ILA network is utilized to implement each next state equation, then the next state forming logic f_{ip} in Equation 1 would be a random logic function in sum of products form. For example, f_{12} would be $y_1\overline{y_2} + y_2y_4$. A circuit to realize the next state forming logic is simply combinational logic and can be formed from NAND gates, NOR gates or from pass transistors.

Some research has been conducted to simplify the next state logic. One such effort was made by Chung-jen Tan [6]. The Tan assignment will produce equations in a form of so called *simple product term equation* in which each coefficient f_{ip} is simplified to an OR expression, instead of a random sum of products expression.

With traditional state assignments, each f_{ip} term in resulting design equations is a function of the partitioning variables of input I_p [7] and the number of partitioning variables

	I_1	I_2	I_3	y_1	y_2	y_3	y_4
A	**A**	B	C	0	0	0	0
B	A	**B**	G	0	0	1	0
C	D	E	**C**	1	0	0	0
D	**D**	F	C	1	1	0	0
E	D	**E**	G	1	0	1	0
F	A	**F**	C	0	1	0	0
G	A	H	**G**	0	1	1	1
H	D	**H**	C	1	1	0	1

Table 1: A flow table and traditional Liu's assignment.

is equal to the number of k-sets in I_p, the maximum number of inputs to f_{ip} logic is equal to the number of k-sets. Therefore, a completely regular ILA circuit for a traditional assignment would contain a k-input next state forming logic for each f_{ip} where k="number of k-sets in input I_p".

A goal of the new design procedure is to minimize the amount of hardware required by the f_{ip} logic. Physically, the minimal form of f_{ip} logic is a wire. The design equation in which all f_{ip} terms are in minimum logic is called a simple term equation.

Definition 1 *A simple term next state equation is a design equation in which each f_{ip} coefficient is a single state variable (or its complement) or a constant.*

In a simple term equation, each coefficient depends on at most *one* state variable. This feature has significant impact on the hardware implementation. With a simple term equation, only a wire is required to generate each f_{ip} coefficient. With a procedure discussed in later sections, the simple term equations for Table 1 can be derived as follows,

$$
\begin{aligned}
Y_1 &= y_1 I_1 + y_3 I_2 + \overline{y_2} I_3 \\
Y_2 &= I_1 \quad + y_4 I_2 + y_2 I_3 \\
Y_3 &= I_1 \quad + y_3 I_2 + 0 \\
Y_4 &= \overline{y_1} I_1 + y_4 I_2 + \overline{y_2} I_3
\end{aligned}
$$

As all coefficients are simple terms, no extra logic is required to generate each f_{ip}.

3.2 New State Assignment

In this research, partition algebra is used to derive simple term equations and synthesize asynchronous circuits. The new state assignment, called η assignment, is proposed to produce the simple term design equations. A relationship between η_i^p partitions and f_{ip} coefficients has been presented in the literature [8]. Theorem 2 in [8] shows part of the discovery and has been used to analyze state assignments in predicting hardware.

Theorem 2 *(1) If $\eta_i^p = \tau_j$, then $f_{ip} = y_j$ or $\overline{y_j}$. (2) If all the states of η_i^p are in one block, then $f_{ip} = 0$ or 1.*

If all f_{ip} of the design equations meet the conditions of Theorem 2, then the coefficients will be simple terms. A sufficient condition of η-partitions for simple term f_{ip} is listed in Corollary 1.

Corollary 1 *If all η_i^p partitions satisfy one of the following conditions:*

1. $\eta_i^p = \tau_j$
2. $\eta_i^p = \{S; \phi\}$
3. $\eta_i^p = \{\phi; S\}$

then the design will yield simple terms, where S is a set of all flow table states.

Proof: The proof follows directly from satisfying Theorem 2.
□

Most STT assignments do not meet the conditions of Corollary 1. However, if an assignment can be generated such that for each η-partition η_j^p under input I_p where η_j^p does not satisfy the conditions from Corollary 1, a new τ-partition τ_k will be created where $\tau_k = \eta_j^p$ to allow $f_{jp} = y_k$ and produce a simple term equation for Y_j.

For example, the flow table shown in Table 1 has eight k-sets:

$$\{ABFG\}, \{AB\}, \{ACDFH\}, \{CDEH\}, \{CE\}, \{DF\}, \{BEG\}, \{GH\}.$$

A set of τ-partitions can be formed to partition each k-set from the rest of states and the results are τ_1 through τ_8 in Table 2 (a). Then from the corresponding η-partitions, it can be found that some η-partitions, such as $\eta_1^2, \eta_3^2, \eta_4^2$ and η_7^2, are not equal to any τ-partitions or a constant. A new τ-partition needs to be formed for each of η-partitions which do not meet the conditions of Corollary 1. For each newly created τ-partition, corresponding η-partitions need to be formed. The new τ-partitions may generate new η-partitions which do not meet any conditions of Corollary 1. Additional τ-partitions would then be required. The partitioning procedure will continue until the conditions of Corollary 1 are met. In the case of assignment for Table 1, $\tau_9, \tau_{10}, \tau_{11}, \tau_{12}$ are formed which in turn generate 12 corresponding η partitions. The results are shown in Table 2 where all η-partitions are equal to a τ-partition or a constant.

Simple term equations can be generated once all η-partitions satisfy Corollary 1. In most cases, however, more τ-partitions (state variables) than necessary have been introduced and can be removed without jeopardizing the simple term feature.

Definition 2 *A τ-partition τ_i is redundant if τ_i and η_i^p for all I_p can be eliminated while the resulting next state equations remain the simple term equations and the state assignment remains a critical race free STT assignment.*

Theorem 3 *In the set of τ-partitions resulting from the η assignment, if a τ_i, which partitions a k-set K_t in I_p from other states is not equal to any η_j^p or its logical complement, where $i \neq j$, then τ_i is redundant.*

$\tau_1 = \{ABFG; CDEH\}$	$\tau_2 = \{AB; CDEFGH\}$	$\tau_3 = \{ACDFH; BEG\}$
$\tau_4 = \{CDEH; ABFG\}$	$\tau_5 = \{CE; ABDFGH\}$	$\tau_6 = \{DF; ABCEGH\}$
$\tau_7 = \{BEG; ACDFH\}$	$\tau_8 = \{GH; ABCDEF\}$	$\tau_9 = \{ABCE; DFGH\}$
$\tau_{10} = \{ABDF; CEGH\}$	$\tau_{11} = \{CEGH; ABDF\}$	$\tau_{12} = \{DFGH; ABCE\}$

(a) τ-partitions

$\eta_1^1 = \{ABFG; CDEH\}$	$\eta_1^2 = \{ABDF; CEGH\}$	$\eta_1^3 = \{BEG; ACDFH\}$
$\eta_2^1 = \{ABFG; CDEH\}$	$\eta_2^2 = \{AB; CDEFGH\}$	$\eta_2^3 = \{-; ABCDEFGH\}$
$\eta_3^1 = \{ABFGCDEH; -\}$	$\eta_3^2 = \{DFGH; ABCE\}$	$\eta_3^3 = \{ACDFH; BEG\}$
$\eta_4^1 = \{CDEH; ABFG\}$	$\eta_4^2 = \{CEGH; ABDF\}$	$\eta_4^3 = \{ACDFH; BEG\}$
$\eta_5^1 = \{-; ABFGCDEH\}$	$\eta_5^2 = \{CE; ABDFGH\}$	$\eta_5^3 = \{ACDFH; BEG\}$
$\eta_6^1 = \{CDEH; ABFG\}$	$\eta_6^2 = \{DF; ABCEGH\}$	$\eta_6^3 = \{-; ABCDEFGH\}$
$\eta_7^1 = \{-; ABFGCDEH\}$	$\eta_7^2 = \{ABCE; DFGH\}$	$\eta_7^3 = \{BEG; ACDFH\}$
$\eta_8^1 = \{-; ABFGCDEH\}$	$\eta_8^2 = \{GH; ABCDEF\}$	$\eta_8^3 = \{BEG; ACDFH\}$
$\eta_9^1 = \{ABFG; CDEH\}$	$\eta_9^2 = \{ABCE; DFGH\}$	$\eta_9^3 = \{ACDFH; BEG\}$
$\eta_{10}^1 = \{ABCDEFGH; -\}$	$\eta_{10}^2 = \{ABDF; CEGH\}$	$\eta_{10}^3 = \{-; ABCDEFGH\}$
$\eta_{11}^1 = \{-; ABCDEFGH\}$	$\eta_{11}^2 = \{CEGH; ABDF\}$	$\eta_{11}^3 = \{ABCDEFGH; -\}$
$\eta_{12}^1 = \{CDEH; ABFG\}$	$\eta_{12}^2 = \{DFGH; ABCE\}$	$\eta_{12}^3 = \{BEG; ACDFH\}$

(b) η-partitions

Table 2: The τ and η partitions

Proof: If the k-set K_t in I_p is the left block of τ_i and the τ_i is not equal to any η_j^p or its logical complement, for all $i \neq j$, then for every $I_q, q \neq p$, there must exist a k-set K_k under I_p in the right block of τ_i such that the stable state of K_t and the stable state of K_k are in the same k-set in I_q. Hence the k-set K_t does not need to be partitioned from k-set K_k for the input transition I_p to I_q. Moreover, the partitioning variable y_i is not needed to generate any simple term equation Y_j. Therefore, τ_i is redundant.
□

For example, in Table 2, τ_2, τ_5, τ_6 and τ_8 do not appear in η-partition set except for $\eta_2^2, \eta_5^2, \eta_6^2$, and η_8^2. Those τ-partitions are redundant according to Theorem 3. Another way of looking at this is that y_2 only appears in the equation for Y_2 and no other - same for y_5, y_6 and y_8. Therefore, they are redundant.

Theorem 4 *If τ_i is a logical complement of τ_j, then y_i can be replaced by $\overline{y_j}$ in all design equations.*

Proof: If τ_i is the logical complement of τ_j, then τ_i and τ_j partition the same k-sets. Only one of them is needed to meet the partitioning condition for a critical race free STT assignment. Moreover, according to the Rule 2 in Theorem 7, all coefficients where $f_{kp} = y_i$ can be replaced by $f_{kp} = \overline{y_j}$.
□

$\tau_1 = \{ABFG; CDEH\}$ $\tau_3 = \{ACDFH; BEG\}$
$\tau_{10} = \{ABDF; CEGH\}$ $\tau_{12} = \{DFGH; ABCE\}$

(a) τ-partitions

$\eta_1^1 = \tau_1$	$\eta_1^2 = \tau_{10}$	$\eta_1^3 = \tau_7$
$\eta_3^1 = 1$	$\eta_3^2 = \tau_{12}$	$\eta_3^3 = \tau_3$
$\eta_{10}^1 = 1$	$\eta_{10}^2 = \tau_{10}$	$\eta_{10}^3 = 0$
$\eta_{12}^1 = \tau_4$	$\eta_{12}^2 = \tau_{12}$	$\eta_{12}^3 = \tau_7$

(b) η-partitions

Table 3: The reduced τ and η partitions

Theorem 4 provides the condition for removing another type of redundancy. For example, τ_4 in Table 2 is the logical complement of τ_1. Therefore, equation Y_4 is redundant equation and all coefficients y_4 in the simple term equations can be replaced by $\overline{y_1}$. Similarly, equation Y_7 is also a redundant equation and all coefficients y_7 in the simple term equations can be replaced by $\overline{y_3}$.

Theorem 4 does not specify which τ-partition should be removed if τ_i is a logical complement of τ_j. The final result may be quite different if τ_i or τ_j is removed because removing such a τ-partition may make more τ-partitions become redundant with the condition of Theorem 3. In order to obtain a better result, the redundancy specified by Theorem 3 first needs to be removed. That will allow the designer to make a better choice by checking fewer τ and η-partitions. For example, τ_9 and τ_{11} become redundant partitions after removing τ_4, τ_7 and corresponding η-partitions $\eta_4^p, \eta_7^p, \forall p = 1, 2, 3$.

By eliminating redundant τ-partitions and η-partitions, the number of partitions is significantly reduced. In the example above, 8 out of 12 τ partitions are removed. Table 3 (a) and (b) show the results.

By using Theorem 3, two stable states under the same input state may have an identical state assignment if they have the same next states under all inputs. To have identical assignment for two stable states is equivalent to merge two corresponding rows in the flow table. However, if the outputs associated with these two stable states are not compatible, such a merging is invalid. A unique state has to be assigned to each stable state so that two outputs can be distinguished.

A partitioning chart is introduced to facilitate state assignment reduction and avoid such invalid merging. The chart has intersection for each pair of k-sets in an input. For each τ-partition τ_i introduced by the state assignment, τ_i will be placed in the intersections where a pair of k-sets are partitioned by τ_i. When a τ_j is removed by the state assignment reduction procedure in accordance with Theorem 3, τ_j must be removed from all intersections in the partitioning chart.

If a blank intersection in a partitioning chart was left once τ_j was removed, then the compatibility of two corresponding rows in the flow table would have to be checked. If

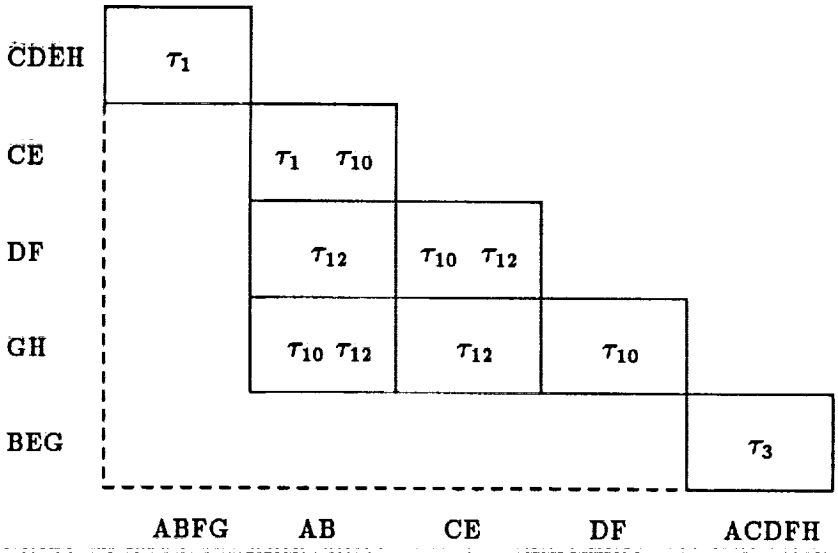

Figure 3: The partitioning chart for the final τ partitions

two outputs are not compatible, τ_j should not be removed even though τ_j is redundant according to Theorem 3. The partitioning chart for the final η assignment is shown in Figure 3. There is at least one τ-partition at each intersection.

The following procedure formalizes the η assignment assignment.

Procedure 1 *η assignment generation:*

Step 1. *Select an initial set of τ-partition τ_i that partition each k-set K_i from the rest of states. Create a partitioning chart for k-sets.*

Step 2. *Generate all η-partitions η_i^p for each input I_p.*

Step 3. *For any η_i^p that do not meet the conditions of Corollary 1, generate a new τ-partition τ_k such that $\tau_k = \eta_i^p$. Add τ_k to the partitioning chart. Return to step 2. The state assignment process is complete when every η_i^p satisfies the conditions of Corollary 1.*

Step 4. *Remove each τ-partition τ_i that meets the conditions of Theorem 3 if there are no merging problem in the partitioning chart by removing τ_i. Also, for each τ_i removed, remove η_i^p under all I_p. Repeat Step 4 until all such τ_i have been eliminated.*

Step 5. *For each pair of τ-partitions τ_i and τ_j that are logical complements, remove τ_i and η-partitions η_i^p under all I_p. Once such a τ_i is removed, return to Step 4.*

Since Procedure 1 involves iterations of step 2 and step 3, it is useful to know if the number of τ-partitions is finite. The following theorem shows closure of Procedure 1.

Theorem 5 *The number of τ-partitions needed to generate any arbitrary η assignment is finite.*

	I_1	I_2	I_3	y_1	y_3	y_{10}	y_{12}
A	**A**	B	C	1	1	1	0
B	A	**B**	G	1	0	1	0
C	D	E	**C**	0	1	0	0
D	**D**	F	C	0	1	1	1
E	D	**E**	G	0	0	0	0
F	A	**F**	C	1	1	1	1
G	A	H	**G**	1	0	0	1
H	D	**H**	C	0	1	0	1

Table 4: The result of η assignment

Proof: If there are k k-sets in a column of I_p, the maximum number of τ-partitions that could be generated is

$$\binom{k}{0} + \binom{k}{1} + \binom{k}{2} + \cdots + \binom{k}{k} = 2^k.$$

This is finite.
□

Corollary 2 *A simple term η assignment exists.*

Proof: A flow table must have at least one input state with more than one k-set, otherwise the circuit is purely combinational. With k k-set ($k \geq 2$), Theorem 5 shows that the maximum of τ-partitions is 2^k. Therefore, an assignment exists because a set of τ-partitions can be constructed.
□

Theorem 6 *The η assignment is a valid STT assignment.*

Proof: A Liu assignment will produce τ-partitions which partition the k-sets. The τ-partitions in this assignment consists of two sets of partitions. The first consists of the initial set of τ-partitions that partitions individual k-sets. The second set consists of the τ-partitions that are created from η-partitions that do not meet conditions of Corollary 1. Hence, both type of τ-partitions are elements of a valid Liu assignment and therefore the overall assignment meets the conditions of an STT assignment.
□

The result of η assignment for flow table Table 1 is given in Table 4.

3.3 Design Equation Generation

With the η assignment, the next design step is to associate a unique state variable y_i with each τ-partition τ_i and determine each f_{ip} term.

Theorem 7 *The next state equations can be derived from the η-partitions with the following roles:*

1. *If $\eta_i^p = \tau_j$, then $f_{ip} = y_j$.*
2. *If the states in the left block of η_i^p are the same as states in the right block of τ_j and vice versa, then $f_{ip} = \overline{y_j}$.*
3. *If all the states of η_i^p are in the left block, then $f_{ip} = 1$.*
4. *If all the states of η_i^p are in the right block, then $f_{ip} = 0$.*

Proof: The proof follows directly from Theorem 2 since the η assignment assigns state variable $y_j = 1$ to states in the left block and $y_j = 0$ to states in the right block of τ-partition τ_j.
□

The following example gives the next state equations by applying Theorem 7 to the result of η assignment in Table 3.

Example 1 *Applying the revised conditions of Theorem 7 to each η-partition η_i^p in Table 3, the next state equations can be derived as follows:*

$$\begin{aligned} Y_1 &= y_1 I_1 + y_{10} I_2 + \overline{y_3} I_3 \\ Y_3 &= I_1 \quad + y_{12} I_2 + y_3 I_3 \\ Y_{10} &= I_1 \quad + y_{10} I_2 + 0 \\ Y_{12} &= \overline{y_1} I_1 + y_{12} I_2 + \overline{y_3} I_3 \end{aligned}$$

The η assignment guarantees to produce the simple term equations in which all coefficients are single variables or constants. The general form for the next state equation is

$$Y_i = \sum_{p=1}^{n} f_{ip} I_p.$$

It has been shown in [10] that this equation can be represented as a pass logic expression. Since only one $I_p = 1$ at a time and the case when all $I_i = 0$ is an undefined situation in an asynchronous flow table, an ILA network will let y_i be passed to Y_i if the term $\overline{I_n}(y_i)$ is added. The equation of Y_i then becomes

$$Y_i = I_1(f_{i1}) + \overline{I_1}(\cdots I_p(f_{ip}) + \overline{I_p}(\cdots (I_n(f_{in}) + \overline{I_n}(y_i)\cdots). \tag{2}$$

The advantage of this equation is to allow Y_i to maintain the same state when all $I_i = 0$, which can happen during an input transition. For example, the simple term equation for Y_1 in Example 1 is

$$Y_1 = y_1 I_1 + y_{10} I_2 + \overline{y_3} I_3$$

Putting this into the form of Equation 2, the expression can then be converted into

$$Y_1 = I_1(y_1) + \overline{I_1}(I_2(y_{10}) + \overline{I_2}(I_3(\overline{y_3}) + \overline{I_3}(y_1)))$$

Similarly, all other equations can be converted to the pass logic expressions in the same way. The results are as follows:

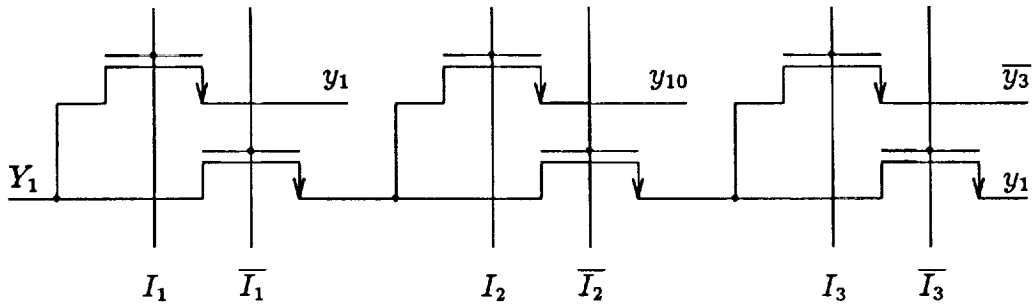

Figure 4: ILA realization of state variable Y_1

$$
\begin{aligned}
Y_1 &= I_1(y_1) + \overline{I_1}(I_2(y_{10}) + \overline{I_2}(I_3(\overline{y_3}) + \overline{I_3}(y_1))) \\
Y_3 &= I_1(1) + \overline{I_1}(I_2(y_{12}) + \overline{I_2}(I_3(y_3) + \overline{I_3}(y_3))) \\
Y_{10} &= I_1(1) + \overline{I_1}(I_2(y_{10}) + \overline{I_2}(I_3(0) + \overline{I_3}(y_{10}))) \\
Y_{12} &= I_1(\overline{y_1}) + \overline{I_1}(I_2(y_{12}) + \overline{I_2}(I_3(\overline{y_3}) + \overline{I_3}(y_{12}))).
\end{aligned}
$$

Obviously, the next state logic f_{ip} is minimized to a wire if the set of next state equations are all simple term equations. It is straightforward then to map the design equations to an ILA network. Figure 4 shows an ILA realization of state variable Y_1.

4 Input and Hazard Characteristics

In addition to the regularity of the ILA network with the η assignment, an ILA realization has other features such as (1) immunity to input 1-1 overlapping and 0-0 crossing, (2) immunity to input bounce while 1-1 overlapping is not present, and (3) free of transition path hazards and input state transition hazards.

Potential conflicts arise in asynchronous sequential networks when more than one input state is present at a time (1-1 overlapping), or when none of the input states are active (0-0 crossing). The overlapping and crossing situations occur because two input states rarely switch at exactly the same time due to differences in delay through forming logic. Most design procedures avoid such uncertainties by setting a constraint either to forbid 1-1 overlapping or to forbid 0-0 crossing of input states. Another form of hazard on an input signal which may cause the circuit to malfunction is the dynamic hazard when input transitions.

Theorem 8 *The ILA architecture tolerates 1-1 input overlapping.*

Proof: In Equation 2, assume the input state I_p have higher priority relative to input state I_q. In the other words, I_p is the control variable of an ILA cell which is closer to the output than the ILA cell with the control variable I_q. If both I_p and I_q are asserted 1, then Y_i will assume value specified by f_{ip} rather than f_{iq}.

In the case where the input switches from I_p to I_q, when I_p is 1, it passes f_{ip} to Y_i and meanwhile cuts off the path of f_{iq} to Y_i, no matter if I_q is set to 1 or 0. With such an

architecture, the 1-1 overlapping of input I_p and I_q has the same effect on the output Y_i as $I_p = 1$, $I_q = 0$.

In the case where the input switches from I_q to I_p, when $I_q = 1$, all Y_i are determined by the f_{iq} values which are propagated through the ILA. Once $I_p = 1$, the f_{ip} values are passed through the ILA independent of whether I_q is 0 or 1. Hence the circuit assumes the proper next state value.

□

For example, in Figure 4, if I_1 and I_2 are active, the circuit for all next state variables will be under the control of I_1 only, and Y_1 will assume the value of y_1. y_{10} can be passed to Y_1 only if $I_1 = 0$ while I_2 is active. Once I_1 is set to 1 again, the value of y_{10} at Y_1 will become y_1 immediately (assuming $\overline{I_1}$ is set to 0 simultaneously).

The 0-0 crossing happens when all input states are 0. Let the input change from I_p to I_q. If input I_p goes to 0 before I_q goes to 1, there will be a period that all input lines are 0. In traditional design, the outputs of the design equations could assume an undeterminant state when all inputs are 0. The ILA architecture solves the problem by providing a path for each state variable to pass y_i to Y_i. When all inputs are 0, it allows Y_i to maintain its current value y_i.

Again, Figure 4 can be used to show the feature of 0-0 crossing tolerance. For example, assuming the input transition is from I_2 to I_3, when $I_2 = 1$, y_{10} is passed to Y_1, and y_1 is fed back into the last ILA cell under $\overline{I_3}$. Once the network is stable, the level of y_{10} is passed to Y_1. When I_2 is set to 0, the path provided by $\overline{I_1}$, $\overline{I_2}$ and $\overline{I_3}$ will still maintain the level of Y_1 at the current value of y_1. The output of the network remains unchanged during the period when all inputs are 0 until I_3 is set to 1. Then a new level of y_7 will be passed to Y_1 and the network assumes a new state.

An input bounce can be considered a dynamic hazard during the transition of input states. With the ability to tolerate input bounce, the ILA network allows extra input transitions to occur before the circuit is stablized. However, the input bounce can be tolerated only if it is not necessary to also tolerate 1-1 overlapping. If an input I_q bounce occurs when two input states I_p and I_q are overlapping, an ambiguity is created regarding the interpreration of the transition; it could be one transition from I_p to I_q or three transitions from I_p to I_q then to I_p back to I_q. In order to avoid the ambiguity, it is assumed that during the input state transition from I_p to I_q the circuit tolerates either the bounce condition or the 1-1 overlapping, but not both.

Theorem 9 *For the simple term equations derived from an η assignment, all partitioning variables under input I_p do not change as the circuit transitions from one state to another under I_p.*

Proof: The η assignment produces the simple term equations of the form

$$Y_i = \cdots + y_i I_p + \cdots$$

where y_i is the partitioning variable of I_p. y_i is the only partitioning variable for all transition paths in the η assignment. According to Theorem 2 in [7], y_i will not change in

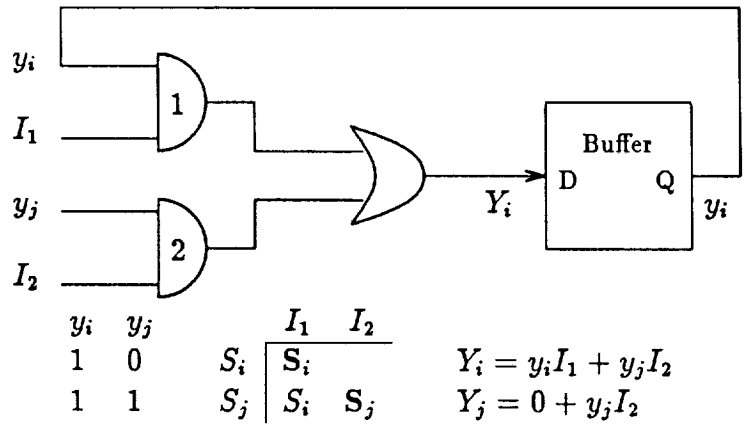

Figure 5: A traditional realization of a partial flow table

any transition under I_p. Hence, the partitioning variables of the simple term equations do not change during the input transitions.
□

From Theorem 9, the partitioning variables do not change when an input signal I_p undergoes a bounce. In the case that a dynamic hazard presents when I_p transitions from 0 to 1, the circuit begins to transition to the new state when I_p goes to 1. Meanwhile, the partitioning variables of I_p remain stable. Since the partitioning variables determine all f_{ip} values [7] and since the partitioning variables are uneffected by the input bounce, the circuit will assume the proper next state value when I_p is stablized.

In the case that a dynamic hazard presents when I_p transitions from 1 to 0, the present state is simply passed to the next state once I_p goes to 0, as this represents a 0-0 crossover condition. The circuit does not transition. When I_p returns to 1 on the bounce condition, again from Theorem 9, there will be no transition in partitioning next state variables. Since the circuit is a function only of the partitioning variables [7], output of the circuit remains unchanged.

A critical race free asynchronous sequential network may still malfunction due to unwanted switching transients in the combinational circuit. A transition path hazard is one that is present within the states of a transition path. The simple term equations for the next state variables have the form

$$Y_i = \cdots + y_i I_p + \cdots$$

where y_i is the partitioning variable of I_p. From Theorem 9, partition variable y_i does not change as the circuit transitions from one state to another in I_p. In general, since the partitioning variables of the simple term equations do not change during the transition, it is impossible for a hazard of any kind to occur. Therefore, there cannot be any static, dynamic and function hazards that occur during the transition between unstable and stable states.

A circuit free of static and dynamic hazard may have a hazard problem caused by the change of more than one variable in the design equation. One type of such multi-variable change hazard is a function hazard. The problem can be illustrated by the partial flow

table in Figure 5 where design equations for Y_i, Y_j and a schematic for Y_i are shown. A hazard exists when an input changes from I_2 to I_1. If the delay of AND gate-1 is longer than the total delays of AND gate-2, the OR gate and the buffer, then the output of gate-1 will remain 0 after the input changes to $I_1 = 1$ and $I_2 = 0$. That will in turn cause y_i to remain at 0 and lock the output of gate-1 to 0. The result is that state variables $y_i y_j$ are set to 00 instead of intended value 10. Moreover, the circuit is locked up in the wrong state until a reset line is provided.

The a problem occurs in a traditional design where a slow gate has the same effect as a 0-0 crossover. Input $I_p = 0$ will set the product term to 0. The ILA circuit solves the problem by maintaining the same state when all inputs go to 0. Also, the 1-1 overlapping problem which may arise in the traditional design due to a slow gate can be tolerated by ILA circuit. The 1-1 overlapping and 0-0 crossover properties of the ILA architecture prevents the circuit from malfunction due to such input state transition hazards.

References

[1] C. Roth, *Fundamentals of Logic Design*, 3rd Ed,. St. Paul, Minn., West Publishing, 1985.

[2] D. Givone, *Introduction to Switching Circuit Theory*, McGraw-Hill, Inc., 1970.

[3] S. Whitaker, "Design of Asynchronous Sequential Circuits Using Pass transistors," Ph.D Dissertation, University of Idaho, Feb. 1988.

[4] S. K. Gopalakrishnan and G. K. Maki, "VLSI Asynchronous Sequential Circuit Design", ICCD, Sept, 1990, pp. 238-242.

[5] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol.24, No.1, Feb. 1989, pp. 71-78.

[6] C. Tan, "State Assignments for Asynchronous Sequential Machines", IEEE Transactions on Computers, Vol. C-20, No. 4, April 1971, pp. 382-391.

[7] S. Gopalakrishnan, G. Kim and G. Maki, "Implications of Tracey's Theorem to Asynchronous Sequential Circuit Design", The 2nd NASA SERC Symposium on VLSI Design, November, 1990, pp. 9.1.1-9.1.11.

[8] G. Maki, D.Sawin and B. Jeng, "Improved State Assignment Selection Tests", IEEE Transactions on Computer, Dec. 1972, pp.1443-1449.

[9] J. Tracey, "Internal State Assignment for Asynchronous Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-15, Aug. 1966, pp. 551-560.

[10] S. K. Gopalakrishnan, "Design of VLSI Asynchronous Sequential Machines," Ph.D Dissertation, University of Idaho, December, 1989.